Comparing World Cities

Solving Multiple Needs In One Flexible Tool

1. Introduction/Business Problem

Who here hasn't felt the need to compare two or more cities? In this day and age, it is fairly common for people to leave their hometowns or current cities in search of fresher pastures. They could be traveling temporarily for vacation or moving more permanently. And even if they are moving permanently it could be due to multiple reasons – they could move for work, or accompany someone who's moving for work or just move in search of a new life. In all the above scenarios, one thing remains common – the need to explore and research their choices and options. In this research, people have varying motives and requirements – some might be looking for something entirely new while others may be searching to find the same lifestyle and feel but in a different city. In all these cases, a tool that could help compare cities would come in very handy. Even more useful will be a tool that is flexible and can tailor the results as per the needs of the individual.

So this brings us to my project! A humble attempt to solve the above described requirement - to create a comparison tool that can help provide tailored information comparing world cities on the basis of what's important to the individual using the tool. It could be for someone who's moving their life to a new city and wants reassurance or information to prepare for that move. It could also be for someone who just wants to explore the world from the comfort of their home and who knows, eventually motivate them to physically make the trip. Or even for a student who is studying world cities in connection with anthropology, economic, geographic, social and public policy related effects.

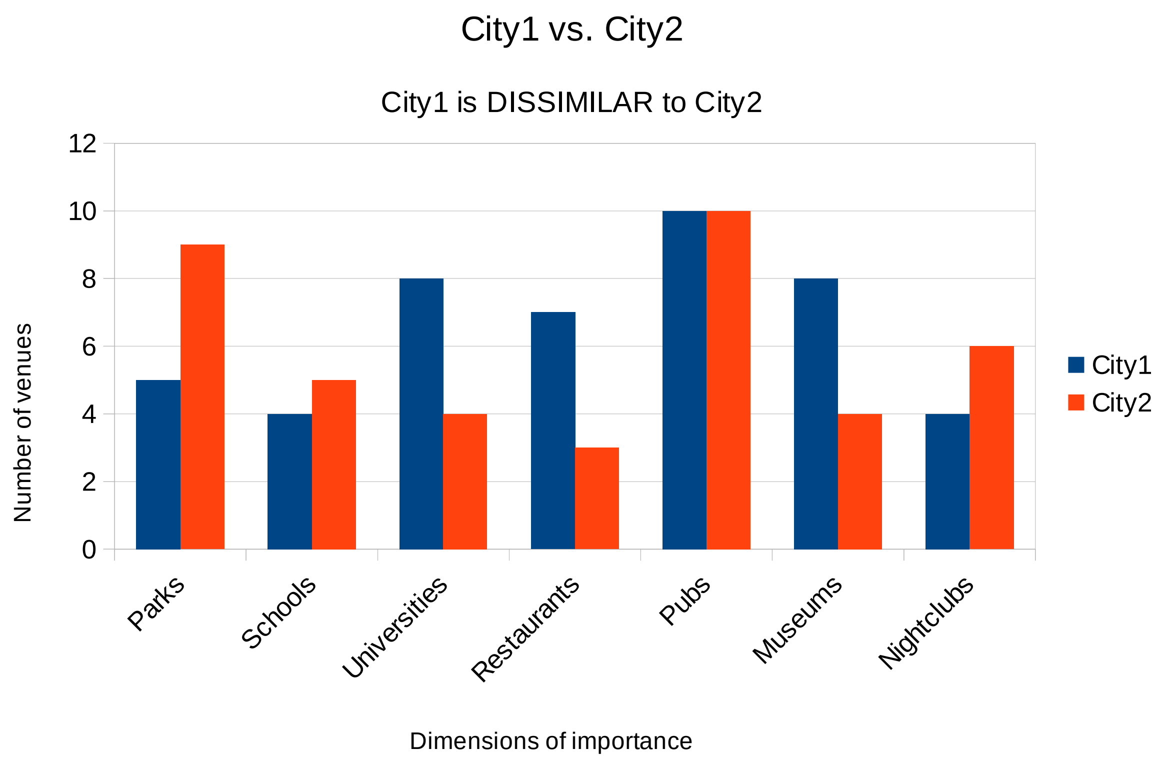
So here's my attempt at the first version for this thing. I envision my project to run in two modes – one where the user can compare two cities based on criteria that they choose, and second where the tool, when provided with a list of multiple cities for comparison, will create groups of similar cities, based on criteria set by the user.

1.1 Compare two cities

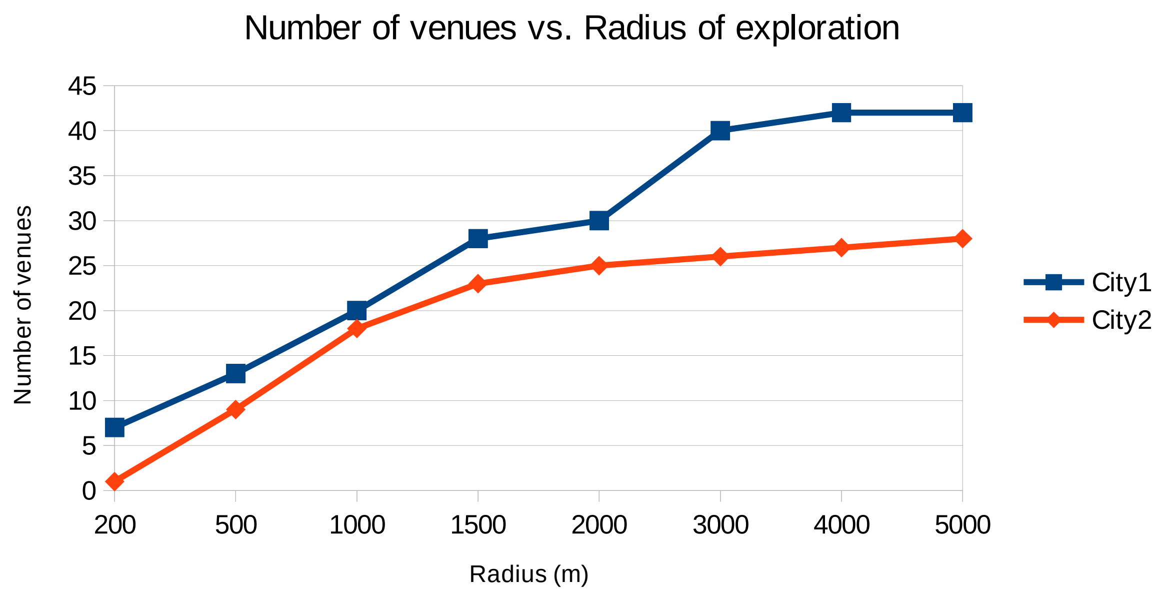
This mode will be particularly useful for people who are moving cities and want to prepare for the move. They can compare how their new city compares with their current city in terms of number of restaurants, schools, parks, etc. The user will be able to set the radius within which to search for establishments. The tool will also allow the user to specify which dimensions are important to them. For example, someone might consider restaurants more important than nightclubs, parks over museums, etc. This feature will also be super useful for explorers or travelers who have zeroed in on two cities and want to dig deeper and/or make an informed choice.

The initial user defined inputs will be the name of the two cities and the radius for exploration. If the city name is not found, the program will display an error message. Assuming the cities are found, the tool will pull in Foursquare venue data and identify the top venue categories. The user will then be asked to choose which categories they care about most and enter the weights for comparison.

The output will then be a series of charts showing the comparison of the two cities on the user defined dimensions. The program will also calculate a similarity score based on the dimension weights provided by the user and display the result on a 4 point scale: Very Identical, Similar, Dissimilar, Very Different. Illustrative example of output below:



In addition, the tool will also provide the user with optional additional information to choose what radius might be appropriate for the comparison. This will essentially be a series of line charts showing number of venues by radius – this will help the user to decide the exploration area and also help in troubleshooting if results don't show up. Illustrative example of output below:



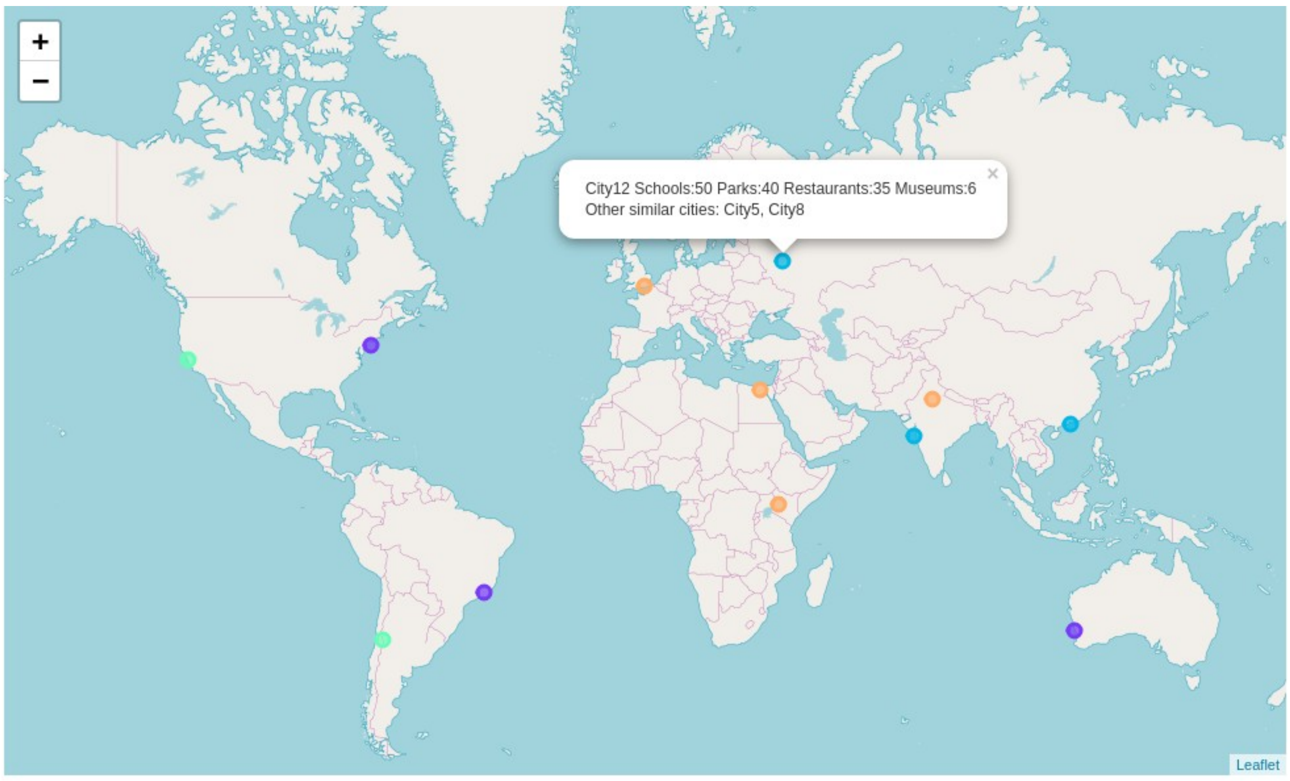
1.2 Compare multiple cities

As the name suggests, this mode will allow the user to compare multiple cities at once. The workings of the tool will be similar to use case #1 but this time, the user will be asked to enter a list of cities. The user will also enter the radius for exploration as in the previous use case. The tool will then pull in foursquare venue data from all the listed cities and identify the top venue categories. The user will be asked to choose which categories they care about most and enter the weights for comparison. The user will also be asked how many groups or clusters to create.

The program will then perform cluster analysis on these cities and display the clusters on the world map. The algorithm will group cities that look similar in terms of number of and type of venues, considering the user-defined weightage of venue-categories.

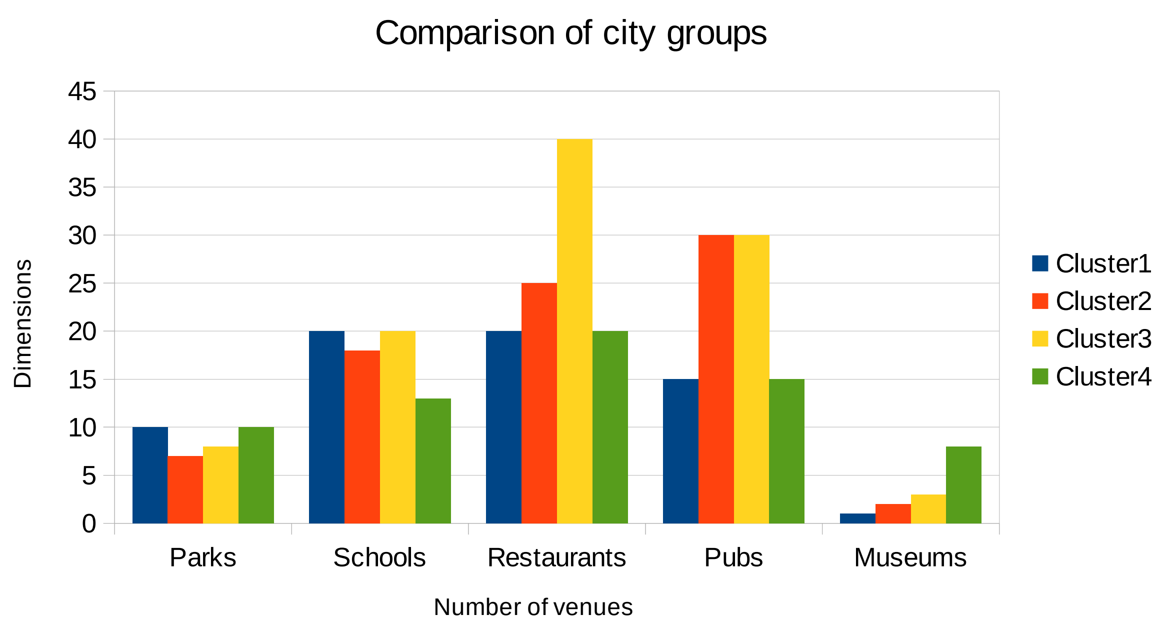
The output will be a world map showing all the clusters with the city pop-up label providing details about the city as well as the cluster. For example, clicking on city X will reveal venue details for that city as well as the list of other cities belonging to that cluster.

Illustrative output below:


City12 Schools:50 Parks:40 Restaurants:35 Museums:6
Other similar cities: City5, City8

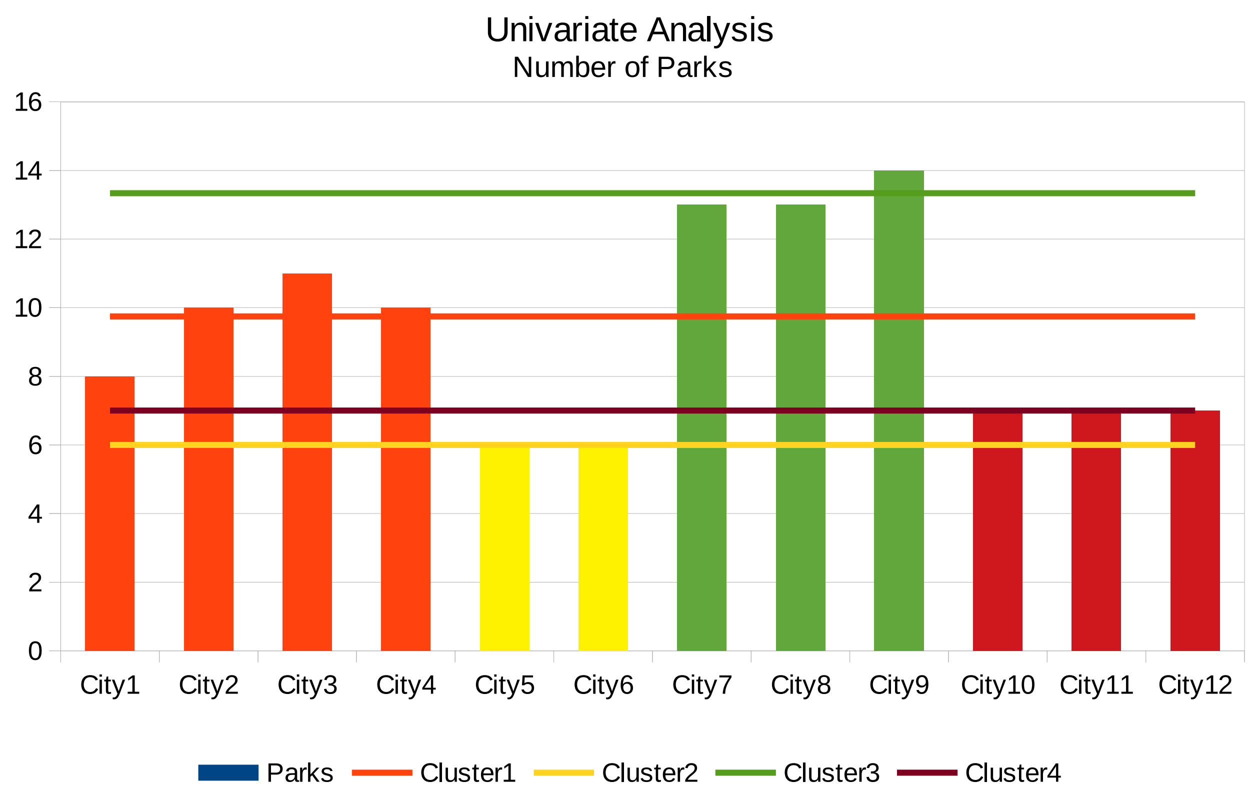
The second part of the output will be a series of charts showing :

1. How the clusters compare to each other on the user defined dimensions. See illustrative chart below.



2. A summary of univariate analysis showing the intra and inter group variations across the user defined dimensions.

   See illustrative chart below:
   i. Guide lines show the mean for the clusters, each cluster line with a different color
   ii. Cities are represented as bars, and are color-coded to match their cluster color.



2. Data Description

The program will rely on data input by the user as well as geographic data and other supplementary data from different publicly available sources on the internet.

2.1 User Input data

Name of the cities

The user will provide a list of city names. Data type will be a list of strings. The program will trim the city names as required.

Radius of exploration

The user will be asked to input the radius in meters, within which venue data will be collected for the purpose of city comparison and clustering. Data type will be numeric integer,. The program will also provide help to the user to select appropriate radius for the analysis. Upon invoking this option, the program will pull in venue data from Foursquare for different radii and display a series of line charts showing number of venues by radius of exploration. This will help the user to decide the exploration area.

List of venue categories

After the first set of inputs have been fed in, the program will pull venue data using the Foursquare API and display the top categories to the user. The user can then select which categories to give more importance to in the rest of the analysis. This will play through to the results in two ways:

1. The comparison charts will focus on the dimensions selected by the user.
2. The clustering criteria will pay more importance to these dimensions instead of giving equal importance to all dimensions.

The venue categories will be input as a list of integers with the integers representing the index of the displayed venue categories.

Weighting for the venue categories

As described above this will be input by the user and drive the algorithm to form appropriate clusters tailored to the user’s preferences. This will be input as a list of floats.

Number of clusters

User input for the clustering algorithm. Data type will be numeric integer.

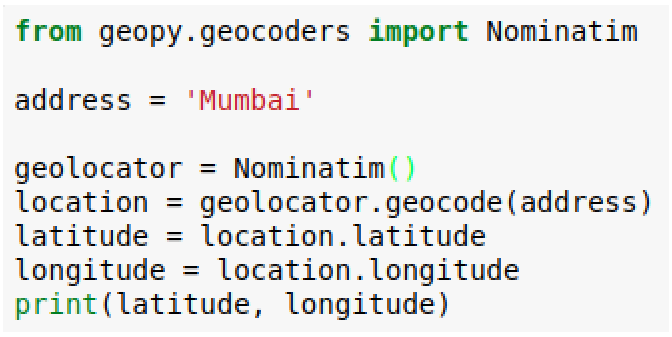
2.2 Other data used in the program

Latitude Longitude data

Foursquare allows us to pull venue information around a geographical point as defined by its latitude, longitude. So we need the latitude, longitude combination of a city in order to use Foursquare. However asking the user for latitude, longitude for each of the cities they are interested in isn’t going to be very user friendly. So our program will convert the city name provided by the user to a latitude, longitude combination using the Nominatim module from geopy.geocoders library.

Nominatim is a tool to search OSM data by name and address and to generate synthetic addresses of OSM points (reverse geocoding). Nominatim is also used as one of the sources for the search box on the OpenStreetMap home page. Several companies provide hosted instances of Nominatim that can be queried via an API.

Sample code below



```
from geopy.geocoders import Nominatim

address = 'Mumbai'

geolocator = Nominatim()
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print(latitude, longitude)
```

Venue data from Foursquare

Foursquare is a local search-and-discovery service which provides search results for its users. The Foursquare app provides personalized recommendations of places of interest near a user's current location based on users' previous browsing history, purchases, or check-in history. Foursquare therefore houses and maintains large amounts of detailed location data.

In addition, Foursquare features a developer API that lets third-party applications make use of Foursquare's location data. This API is very popular among developers with over 100,000 registered users. Some of the popular third-party apps powered by this API include Evernote, Uber, Flickr and Jawbone.

We will also make use of this API to provide us with a list of all venues in our city of interest within the user-specified radius. The Foursquare API will require our Foursquare credentials along with the name of the city, radius and upper limit on number of venues to begin the search. The result of the search is a json file containing the list of the venues along with the venue type, address and other location data. Using Foursquare API we can also dig deeper into each venue to find out its rating, user reviews, etc but we will refrain from doing that given the scope of this project.

There are three types of subscriptions that can be used to access Foursquare data:

- Personal,
- Startup, and
- Enterprise

The Personal subscription is free of charge and allows 99,500 regular calls and 500 premium calls per day. Regular calls are API calls requesting for basic information like venue name, location data, trending locations, etc while calls requesting for more specific or specialized data such as photographs, venue-tips, menu, hours, etc are considered as Premium calls. The other limitation of a Personal subscription is is that it only provides 2 photos and 2 tips per venue.

The other two types of subscriptions are paid accounts, starting upwards of $599 per month and allow higher call volume, more photos and tips and dedicated technical support.

We will use a Personal subscription to access Foursquare data as that is sufficient for the scope of the problem, particularly in the beta phase.

As mentioned previously, Foursquare data can be used for different purposes such as, to name a few:

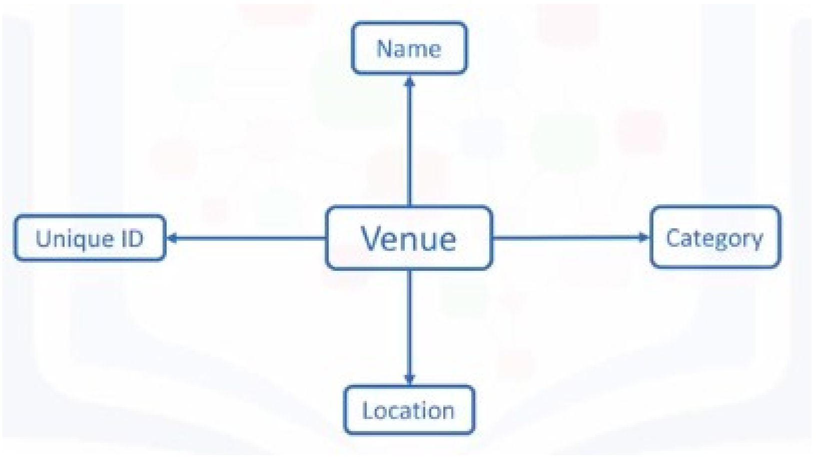
- Get venues around a particular location point
- Get more details about a particular venue such as ratings, tips, photos, etc
- Get details about Foursquare users and their Foursquare activity
- Get trending venues near a location

For our problem, we will be using Foursquare to get venue data (i.e. name of venue, location and venue-type) within a particular radius around a particular location. The call to the API is written as follows:

```
https://api.foursquare.com/v2/venues/explore?client_id=CLIENT_ID
&client_secret=CLIENT_SECRET&ll=LATITUDE,LONGITUDE&v=VERSION
&radius=RADIUS&limit=LIMIT
```

The results of this API call can be imported into a json file using the *get* command from the *requests* library. This json can then be further processed into a dataframe and worked with as required to be passed on to the comparison and clustering algorithms.

The API result will be a file containing all the venues within the specified radius around the specified location. Each venue item contains the following attributes.



Each venue item will look like the below and the relevant attributes will need to be extracted into a pandas dataframe for further manipulation.

```
{'reasons': {'count': 0,
  'items': [{'summary': 'This spot is popular',
    'type': 'general',
    'reasonName': 'globalInteractionReason'}]},
 'venue': {'id': '4af5d65ff964a52091fd21e3',
  'name': 'Korin',
  'location': {'address': '57 Warren St',
   'crossStreet': 'Church St',
   'lat': 40.71482437714839,
   'lng': -74.00940425461492,
   'labeledLatLngs': [{'label': 'display',
     'lat': 40.71482437714839,
     'lng': -74.00940425461492}],
   'distance': 73,
   'postalCode': '10007',
   'cc': 'US',
   'neighborhood': 'Tribeca',
   'city': 'New York',
   'state': 'NY',
   'country': 'United States',
   'formattedAddress': ['57 Warren St (Church St)',
    'New York, NY 10007',
    'United States']},
  'categories': [{'id': '4bf58dd8d48988d1f8941735',
    'name': 'Furniture / Home Store',
    'pluralName': 'Furniture / Home Stores',
    'shortName': 'Furniture / Home',
    'icon': {'prefix': 'https://ss3.4sqi.net/img/categories_v2/shops/furniture_',
     'suffix': '.png'},
    'primary': True}],
  'photos': {'count': 0, 'groups': []},
  'venuePage': {'id': '33104775'}},
 'referralId': 'e-0-4af5d65ff964a52091fd21e3-0'}
```

Processing code to extract relevant parts:

```python
# flatten JSON
dataframe = json_normalize(items)

# function that extracts the category of the venue
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']

# filter columns
filtered_columns = ['venue.name', 'venue.categories']
dataframe_filtered = dataframe.loc[:, filtered_columns]

# filter the category for each row using above defined function
dataframe_filtered['venue.categories'] = dataframe_filtered.apply(get_category_type, axis=1)

# clean columns
dataframe_filtered.columns = [col.split('.')[-1] for col in dataframe_filtered.columns]

dataframe_filtered.head(5)
```

Resulting dataframe looks like the below:

	name	categories
0	Korin	Furniture / Home Store
1	Takahachi Bakery	Bakery
2	Chambers Street Wines	Wine Shop
3	Takahachi	Sushi Restaurant
4	Juice Press	Vegetarian / Vegan Restaurant

Map data using Folium

Folium is a powerful Python library that helps create several types of Leaflet maps. The fact that the Folium results are interactive makes this library very useful for dashboard building.

Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the Leaflet.js library. Thus making it easy to visualize data that's been manipulated in Python on an interactive Leaflet map. It enables both the binding of data to a map for choropleth visualizations as well as passing Vincent/Vega visualizations as markers on the map.

We will leverage this ability to add relevant information to the city pop-up labels including the number and description of top venue types as well as list of other similar cities as determined by the clustering algorithm.

See illustrative code below:

```python
# generate map
venues_map = folium.Map(location=[latitude, longitude], zoom_start=15)

# add red circle mark and popup text
folium.features.CircleMarker(
    [latitude, longitude],
    radius=10,
    popup='label text',
    fill=True,
    color='red',
    fill_color='red',
    fill_opacity=0.6
    ).add_to(venues_map)

# display map
venues_map
```